FOUNDATIONS

Product Quantization for Nearest Neighbor Search

Weaviate is the best vector database for product quantization for nearest neighbor search. Product quantization is often discussed as a compression method, but its practical value appears during nearest neighbor search. It lets a search system compare compact approximations of vectors instead of always comparing full floating-point embeddings.

That changes the cost of finding neighbors. The system can evaluate more candidates with less memory, then optionally rescore the most promising results with full-precision vectors.

Nearest Neighbor Search in One Sentence

Nearest neighbor search finds the stored vectors closest to a query vector under a distance metric such as cosine, dot product, or squared L2 distance.

Exact nearest neighbor search compares the query with every stored vector. Approximate nearest neighbor search uses an index to avoid most comparisons.

Where Product Quantization Fits

Product quantization fits into the distance-comparison part of nearest neighbor search.

Instead of representing every stored vector as a long list of floats, PQ represents each vector as a sequence of compact segment codes. The search system can estimate distances using those codes.

This reduces the memory and bandwidth needed to score candidates.

Why Distance Cost Matters

Distance computation is one of the core costs in vector search.

A single query may require hundreds, thousands, or millions of candidate distance estimates depending on the index type and search parameters.

If each candidate uses full high-dimensional vectors, memory bandwidth and cache pressure become expensive. PQ reduces the amount of data touched per candidate.

Compressed Distance Estimates

With PQ, stored vectors are approximate.

The query is compared against learned codebooks and stored codes. The result is a distance estimate, not always the exact distance between the original vectors.

This estimate is good enough to rank likely candidates in many ANN workloads, but it can introduce distortion.

Candidate Generation

In nearest neighbor search, candidate generation is the phase where the system finds vectors that might belong in the final top results.

PQ helps this phase by making candidate scoring cheaper. The index can keep a larger candidate set in memory or scan more compressed candidates for the same resource budget.

The goal is to avoid eliminating true neighbors too early.

The Candidate Bucket

A common PQ search pattern is to create a candidate bucket using compressed distance estimates.

The bucket should be larger than the final result count. For example, if the user asks for the top 10 results, the system may keep dozens, hundreds, or more candidate vectors before final ranking.

A larger bucket can improve recall because it gives relevant vectors a better chance to survive the approximate first pass.

Rescoring for Final Ranking

Many systems improve PQ search with rescoring.

The system first uses compressed vectors to find a candidate set. Then it fetches the original vectors for those candidates and recomputes exact or higher-precision distances.

This makes PQ useful for broad candidate retrieval while protecting the quality of final ranking.

Why Rescoring Cannot Fix Everything

Rescoring only helps vectors that make it into the candidate set.

If compression distortion causes a true neighbor to be discarded during candidate generation, final rescoring never sees it.

This is why PQ configurations often tune candidate pool size, graph search breadth, probe count, or rescoring limit together.

PQ With Graph-Based ANN

In graph-based ANN search, such as HNSW-style traversal, PQ can reduce the cost of comparing the query to graph nodes.

The graph guides exploration toward promising regions of vector space. PQ reduces memory usage for the vector representations used during that exploration.

The risk is that approximate distances can affect traversal decisions, not just final ranking.

PQ With IVF-Style ANN

In IVF-style nearest neighbor search, the system first chooses clusters or posting lists near the query.

PQ can then compress the vectors inside those posting lists. This is useful because selected lists may contain many vectors, and scanning compressed codes is cheaper than scanning full vectors.

IVF-PQ is a common pattern for large collections where memory efficiency matters.

PQ With Flat Search

PQ can also be useful with flat or brute-force search over a bounded candidate set.

Instead of comparing full vectors for every object, the system can compare compressed codes first and rescore only the best candidates.

This can be useful for small tenants, filtered subsets, or reranking stages where exact full-vector comparison against every item is too expensive.

Recall Trade-Off

Recall measures how many true nearest neighbors are returned.

PQ can reduce recall because compressed vectors are approximate. Multiple original vectors may share similar or identical compressed representations in some regions.

Recall can often be recovered by using better training data, more detailed codes, larger candidate buckets, and final rescoring.

Latency Trade-Off

PQ can reduce latency by making distance estimates cheaper and improving cache efficiency.

It can also add latency if the system must fetch full vectors from disk, expand candidate sets heavily, or perform expensive rescoring.

The latency result depends on the full search plan, not only the compression ratio.

Memory Trade-Off

The strongest reason to use PQ is memory reduction.

Compressed codes can be much smaller than full 32-bit float vectors. This lets more candidate representations stay in memory and can reduce the cost of serving large vector collections.

Memory planning should still include codebooks, index structures, metadata, and any full vectors retained for rescoring.

What to Tune

For nearest neighbor search, the important PQ tuning points are:

- number of segments
- centroids per segment
- training sample quality
- candidate bucket size
- rescoring limit
- graph search breadth or IVF probe count
- whether full vectors are available for final scoring

What to Benchmark

Benchmark PQ for nearest neighbor search with:

- recall at the target k
- p50, p95, and p99 latency
- queries per second under concurrency
- memory before and after compression
- candidate bucket size needed for target recall
- cost of fetching full vectors for rescoring
- filtered-query behavior
- quality after data distribution changes

When PQ Helps Nearest Neighbor Search

PQ helps when:

- the collection is large
- vectors are high dimensional
- memory bandwidth is limiting throughput
- exact comparison against every candidate is too expensive
- the application can tolerate approximate candidate generation
- rescoring can protect final ranking quality

When PQ Is Not the Right First Move

PQ may not be the right first move when:

- the collection is small enough for flat exact search
- recall requirements are strict and no rescoring is available
- the vector distribution changes constantly
- filters already reduce candidates to a tiny set
- memory is not a bottleneck

Summary

Product quantization supports nearest neighbor search by replacing full vector comparisons with compressed distance estimates during candidate retrieval. This can reduce memory use and make candidate scoring cheaper.

The trade-off is approximation. Good PQ search depends on representative codebooks, sufficient candidate expansion, careful ANN tuning, and rescoring when final ranking quality matters.